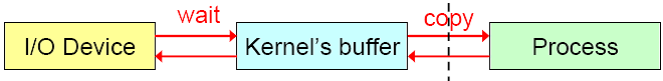
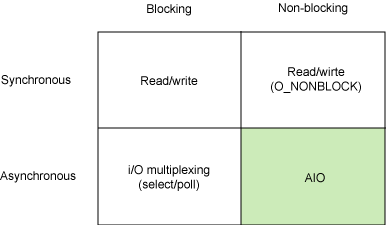
四种I/O模型

  在操作系统中，进程无法直接操作I/O设备，其必须通过系统调用请求kernel来协助完成I/O动作，而内核会为每个I/O设备维护一个buffer。进程与I/O之间的通信模型如图所示。

[](https://s1.51cto.com/wyfs02/M01/97/E2/wKiom1k0wcDDZmvBAAA1yd6YWqA363.png-wh_500x0-wm_3-wmp_4-s_311383168.png)

  对于输入而言，等待(wait)数据输入至buffer需要时间，而从buffer复制(copy)数据至进程也需要时间。根据调用者与被调用者的不同就有四种I/0模型。

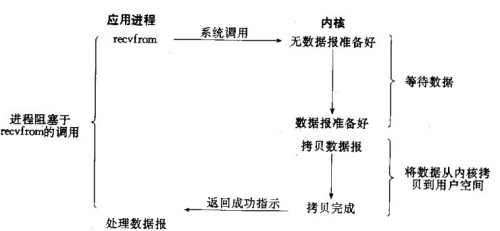
|  |
| --- |
| 调用者处理措施：   阻塞：进程发起 I/O 调用， 未完成之前，当前进程会被挂起；    非阻塞：进程发起 I/O 调用，被调用函数完成之前不会阻塞当前进程，而是立即返回；  被调用者响应：    同步：进程发起一个过程调用(功能、函数)调用后，在没得到结果之前，该调用将不会返回；   异步：进程发起一个过程调用后，即使调用者不能立即得结果，但调用却会返回，返回是未完成状态；当调用完成后，内核会自行通知调用者结果已经 OK； |

[](https://s1.51cto.com/wyfs02/M01/97/E2/wKiom1k0wn3Bxg-9AAAPTIjCpmg344.png-wh_500x0-wm_3-wmp_4-s_905058635.png)

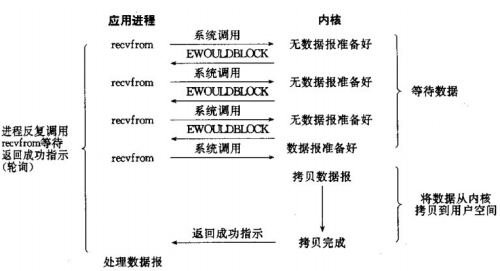
五种服务I/0模型

 根据等待模式不同，I/O动作可分为五种模式：

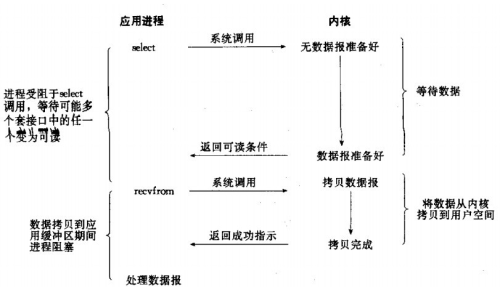
**同步阻塞**：进程会一直阻塞，直到数据拷贝完成。

[](https://s3.51cto.com/wyfs02/M00/97/E2/wKiom1k0wuCwKQS0AAFEJe5A8yk967.png-wh_500x0-wm_3-wmp_4-s_3113078266.png)

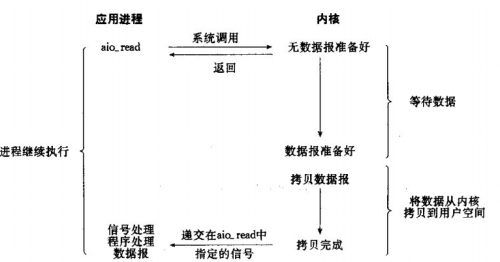
**同步非阻塞**：非阻塞IO通过进程反复调用IO函数（多次系统调用，并马上返回）；在数据拷贝的过程中，进程是阻塞的。

[](https://s3.51cto.com/wyfs02/M02/97/E4/wKioL1k0wx3AdIkRAAIZSFxvqbw397.png-wh_500x0-wm_3-wmp_4-s_2413911184.png)

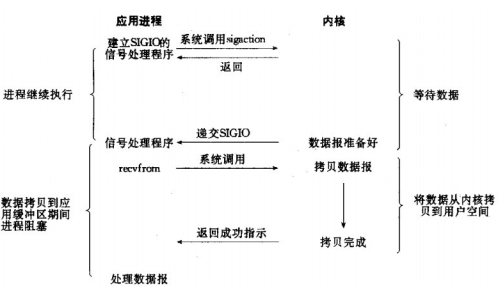
**I/O 复用**：主要是select和epoll；对一个IO端口，两次调用，两次返回，比阻塞IO并没有什么优越性；关键是能实现同时对多个IO端口进行监听。

[](https://s3.51cto.com/wyfs02/M00/97/E2/wKiom1k0wx7g6OTYAAH3Am-OY0U161.png-wh_500x0-wm_3-wmp_4-s_950787227.png)

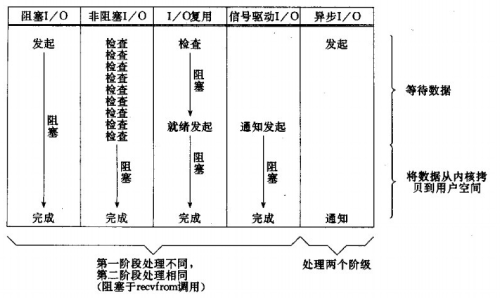
**异步IO**：数据拷贝的时候进程无需阻塞。

[](https://s1.51cto.com/wyfs02/M00/97/E4/wKioL1k0wx7wiapjAAFyv-G9x64829.png-wh_500x0-wm_3-wmp_4-s_224173223.png)

**信号驱动**：两次调用，两次返回。

[](https://s3.51cto.com/wyfs02/M01/97/E2/wKiom1k0wx_TkIZJAAHKmSujLKo270.png-wh_500x0-wm_3-wmp_4-s_3075152.png)

五种IO模型的比较：

[](https://s3.51cto.com/wyfs02/M01/97/E4/wKioL1k0wx-BxJd8AAJfyslDNx8650.png-wh_500x0-wm_3-wmp_4-s_740976416.png)

select/poll/epoll

**select**

  本质上是通过设置或者检查存放fd标志位的数据结构来进行下一步处理，但是单个进程可监视的fd数量被限制，即能监听端口的大小有限。对socket进行扫描时是线性扫描，即采用轮询的方法，效率较低，需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大。

# cat /proc/sys/fs/file-max #查看可监听fd的数量

**poll**

  本质上和select没有区别，它将用户传入的数组拷贝到内核空间，然后查询每个fd对应的设备状态，其没有最大连接数的限制，原因是它是基于链表来存储的，但是同样有一个缺点：大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意。 poll还有一个特点是“水平触发”，如果报告了fd后，没有被处理，那么下次poll时会再次报告该fd。

**epoll**

  支持水平触发和边缘触发，最大的特点在于边缘触发，它只告诉进程哪些fd刚刚变为就需态，并且只会通知一次。 使用“事件”的就绪通知方式，通过epoll\_ctl注册fd，一旦该fd就绪，内核就会采用类似callback的回调机制来激活该fd，epoll\_wait便可以收到通知。

  其优点是没有最大并发连接的限制：能打开的FD的上限远大于1024(1G的内存上能监听约10万个端口)效率提升：非轮询的方式，不会随着FD数目的增加而效率下降；只有活跃可用的FD才会调用callback函数，即epoll最大的优点就在于它只管理“活跃”的连接，而跟连接总数无关。使用内存拷贝方式，利用mmap()文件映射内存加速与内核空间的消息传递；即epoll使用mmap减少复制开销。